Beginning with C++

2

Key Concepts

C with classes | C++ features | Main function | C++ comments | Output operator | Input operator | Header file | Return statement | Namespace | Variables | Cascading of operators | C++ program structure | Client-server model | Source file creation | Compilation | Linking

2.1 What is C++?

C++ is an object-oriented programming language. It was developed by Bjarne Stroustrup at AT&T Bell Laboratories in Murray Hill, New Jersey, USA, in the early 1980's. Stroustrup, an admirer of Simula67 and a strong supporter of C, wanted to combine the best of both the languages and create a more powerful language that could support object-oriented programming features and still retain the power and elegance of C. The result was C++. Therefore, C++ is an extension of C with a major addition of the class construct feature of Simula67. Since the class was a major addition to the original C language, Stroustrup initially called the new language 'C with classes'. However, later in 1983, the name was changed to C++. The idea of C++ comes from the C increment operator ++, thereby suggesting that C++ is an augmented (incremented) version of C.

During the early 1990's the language underwent a number of improvements and changes. In November 1997, the ANSI/ISO standards committee standardised these changes and added several new features to the language specifications.

C++ is a superset of C. Most of what we already know about C applies to C++ also. Therefore, almost all C programs are also C++ programs. However, there are a few minor differences that will prevent a C program to run under C++ compiler. We shall see these differences later as and when they are encountered.

The most important facilities that C++ adds on to C are classes, inheritance, function overloading, and operator overloading. These features enable creation of abstract data types, inherit properties from existing data types and support polymorphism, thereby making C++ a truly object-oriented language.

The object-oriented features in C++ allow programmers to build large programs with clarity, extensibility and ease of maintenance, incorporating the spirit and efficiency of C. The addition of new features has transformed C from a language that currently facilitates top-down, structured design, to one that provides bottom-up, object-oriented design.

2.2 Applications of C++

C++ is a versatile language for handling very large programs. It is suitable for virtually any programming task including development of editors, compilers, databases, communication systems and any complex real-life application systems.

- Since C++ allows us to create hierarchy-related objects, we can build special object-oriented libraries which can be used later by many programmers.
- While C++ is able to map the real-world problem properly, the C part of C++ gives the language the ability to get close to the machine-level details.
- C++ programs are easily maintainable and expandable. When a new feature needs to be implemented, it is very easy to add to the existing structure of an object.
- It is expected that C++ will replace C as a general-purpose language in the near future.

2.3 A Simple C++ Program

Let us begin with a simple example of a C++ program that prints a string on the screen.

Program 2.1 Printing A String

```
#include <iostream> // include header file
using namespace std;
int main()
{

  cout << "C++ is better than C.\n"; // C++ statement

  return 0;
}      //End of example
```

This simple program demonstrates several C++ features.

Program Features

Like C, the C++ program is a collection of functions. The above example contains only one function, **main().** As usual, execution begins at main(). Every C++ program must have a **main().** C++ is a free-form language. With a few exceptions, the compiler ignores carriage returns and white spaces. Like C, the C++ statements terminate with semicolons.

Comments

C++ introduces a new comment symbol // (double slash). Comments start with a double slash symbol and terminate at the end of the line. A comment may start anywhere in the line, and whatever follows till the end of the line is ignored. Note that there is no closing symbol.

The double slash comment is basically a single line comment. Multiline comments can be written as follows:

```
// This is an example of
// C++ program to illustrate
// Some of its features
```

The C comment symbols /*, */ are still valid and are more suitable for multiline comments. The following comment is allowed:

```
/* This is an example of
   C++ program to illustrate
   some of its features
*/
```

We can use either or both styles in our programs. Since this is a book on C++, we will use only the C++ style. However, remember that we can not insert a // style comment within the text of a program line. For example, the double slash comment cannot be used in the manner as shown below:

```
for(j=0; j<n; /* loops n times */ j++)
```

Output Operator

The only statement in Program 2.1 is an output statement. The statement

```
cout<< "C++ is better than C."
```

causes the string in quotation marks to be displayed on the screen. This statement introduces two new C++ features, cout and <<. The identifier cout (pronounced as ‘C out’) is a predefined object that represents the standard output stream in C++. Here, the standard output stream represents the screen. It is also possible to redirect the output to other output devices. We shall later discuss streams in detail.

The operator << is called the *insertion or put to* operator. It inserts (or sends) the contents of the variable on its right to the object on its left (Fig 2.1).

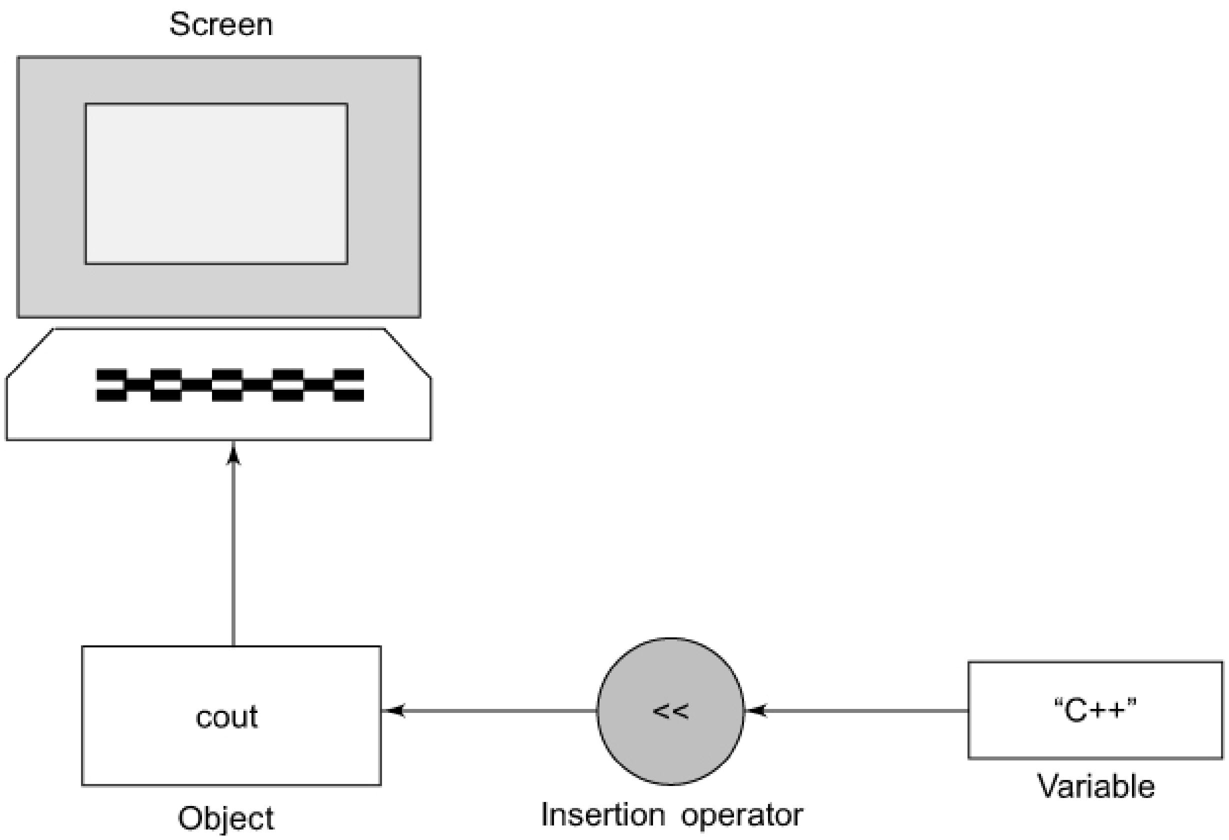

Fig. 2.1 *Output using insertion operator*

The object **cout** has a simple interface. If string represents a string variable, then the following statement will display its contents:

```
cout << string;
```

You may recall that the operator << is the bit-wise left-shift operator and it can still be used for this purpose. This is an example of how one operator can be used for different purposes, depending on the context. This concept is known as *operator overloading,* an important aspect of polymorphism. Operator overloading is discussed in detail in Chapter 7.

It is important to note that we can still use printf() for displaying an output. C++ accepts this notation. However, we will use cout << to maintain the spirit of C++.

The iostream File

A C++ program typically contains pre-processor directive statements at the beginning. Such statements are preceded with a # symbol to indicate the presence of a pre-processor directive to the compiler, which in turn lets the pre-processor handle the # directive statement. All C++ programs begin with a #include directive that includes the specified header file contents into the main program.

We have used the following #include directive in the program:

```
#include <iostream>
```

This directive causes the preprocessor to add the contents of the iostream file to the program. It contains declarations for the identifier **cout** and the operator <<. Some old versions of C++ use a header file called iostream.h. This is one of the changes introduced by ANSI C++. (We should use iostream.h if the compiler does not support ANSI C++ features.)

The header file **iostream** should be included at the beginning of all programs that use input/output statements. Note that the naming conventions for header files may vary. Some implementations use **iostream.hpp;** yet others **iostream.hxx.** We must include appropriate header files depending on the contents of the program and implementation.

Tables 2.1 and 2.2 provide lists of C++ standard library header files that may be needed in C++ programs. The header files with .h extension are “old style” files which should be used with old

Table 2.1 *Commonly used old-style header files*

Header file	*Contents and purpose*	*New version*
<assert.h>	Contains macros and information for adding diagnostics that aid program debugging	<cassert>
<ctype.h>	Contains function prototypes for functions that test characters for certain properties, and function prototypes for functions that can be used to convert lowercase letters to uppercase letters and vice versa.	<cctype>
<float.h>	Contains the floating-point size limits of the system.	<cfloat>
<limits.h>	Contains the integral size limits of the system.	<climits>
<math.h>	Contains function prototypes for math library functions.	<cmath>
<stdio.h>	Contains function prototypes for the standard input/output library functions and information used by them.	<cstdio>
<stdlib.h>	Contains function prototypes for conversion of numbers to text, text to numbers, memory allocation, random numbers, and various other utility functions.	<cstdlib>
<string.h>	Contains function prototypes for C-style string processing functions.	<cstring>
<time.h>	Contains function prototypes and types for manipulating the time and date.	
<iostream.h>	Contains function prototypes for the standard input and standard output functions.	<iostream>
<iomanip.h>	Contains function prototypes for the stream manipulators that enable formatting of streams of data.	<iomanip>
<fstream.h>	Contains function prototypes for functions that perform input from files on disk and output to files on disk.	<fstream>

Table 2.2 *New header files included in ANSI C++*

Header file	*Contents and purpose*
<utility>	Contains classes and functions that are used by many standard library header files.
<vector>, <list>, <deque> <queue>, <set>, <map>, <stack>, <bitset>	The header files contain classes that implement the standard library containers. Containers store data during a program's execution. We discuss these header files in Chapter 14.
<functional>	Contains classes and functions used by algorithms of the standard library.
<memory>	Contains classes and functions used by the standard library to allocate memory to the standard library containers.
<iterator>	Contains classes for manipulating data in the standard library containers.
<algorithm>	Contains functions for manipulating data in the standard library containers.
<exception>, <stdex-cept>	These header files contain classes that are used for exception handling.
<string>	Contains the definition of class string from the standard library. Discussed in Chapter 15
<sstream>	Contains function prototypes for functions that perform input from strings in memory and output to strings in memory.
<locale>	Contains classes and functions normally used by stream processing to process data in the natural form for different languages (e.g., monetary formats, sorting strings, character presentation, etc.)
<limits>	Contains a class for defining the numerical data type limits on each computer platform.
<typeinfo>	Contains classes for run-time type identification (determining data types at execution time).

compilers. Table 2.1 also gives the version of these files that should be used with the ANSI standard compilers.

Apart from #include, the other pre-processor directives such as #define, #error, etc., work the same way as they would in C. But, their usage in C++ is quite limited due to the availability of other means for achieving the same functionality.

Namespace

Namespace is a new concept introduced by the ANSI C++ standards committee. This defines a scope for the identifiers that are used in a program. For using the identifiers defined in the **namespace** scope we must include the using directive, like

```
using namespace std;
```

Here, **std** is the namespace where ANSI C++ standard class libraries are defined. All ANSI C++ programs must include this directive. This will bring all the identifiers defined in **std** to the current global scope. **using** and **namespace** are the new keywords of C++. Namespaces are discussed in detail in Chapter 16.

Return Type of main( )

In C++, main() returns an integer type value to the operating system. Therefore, every main() in C++ should end with a return(0) statement; otherwise a warning or an error might occur. Since **main()** returns an integer type value, return type for main() is explicitly specified as **int.** Note that the default return type for all functions in C++ is **int.** The following **main** without type and return will run with a warning:

```
main()
{
  ......
  ......
}
```

2.4 More C++ Statements

Let us consider a slightly more complex C++ program. Assume that we would like to read two numbers from the keyboard and display their average on the screen. C++ statements to accomplish this is shown in Program 2.2.

Program 2.2 Average of Two Numbers

```
#include <iostream>

using namespace std;

int main()
 {

      float numberl, number2, sum, average;
      cout << "Enter two numbers: "; // prompt

      cin >> numberl; // Reads numbers
      cin >> number2; // from keyboard

      sum = numberl + number2;
      average = sum/2;

      cout << "Sum = " << sum << "\n";
      cout << "Average = " << average << "\n";

      return 0;
 }
```

The output of Program 2.2 would be:

```
Enter two numbers: 6.5 7.5
Sum = 14
Average = 7
```

Variables

The program uses four variables number1, number2, sum, and average. They are declared as type float by the statement.

```
float numberl, number2, sum, average;
```

All variables must be declared before they are used in the program.

Input Operator

The statement

```
cin >> numberl;
```

is an input statement and causes the program to wait for the user to type in a number. The number keyed in is placed in the variable number1. The identifier **cin** (pronounced 'C in') is a predefined object in C++ that corresponds to the standard input stream. Here, this stream represents the keyboard.

The operator >> is known as *extraction or get from* operator. It extracts (or takes) the value from the keyboard and assigns it to the variable on its right (Fig 2.2). This corresponds to the familiar scanf() operation. Like <<, the operator >> can also be overloaded.

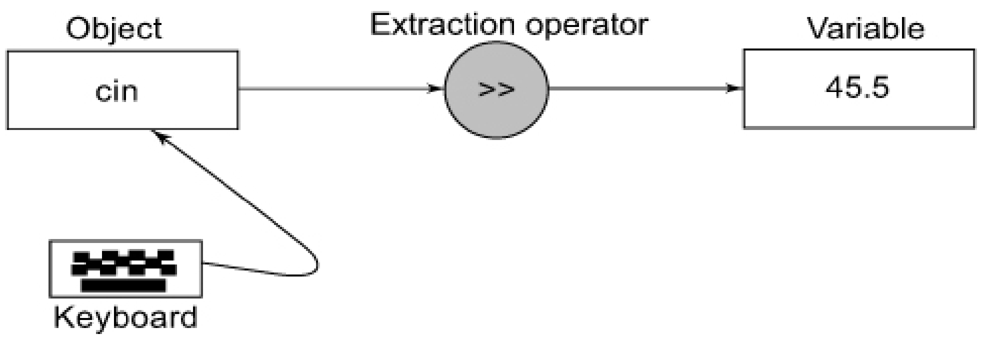

Fig. 2.2 *Input using extraction operator*

Cascading of I/O Operators

We have used the *insertion operator*<< repeatedly in the last two statements for printing results.

The statement

```
cout << "Sum = " << sum << "\n";
```

first sends the string “Sum =“ to cout and then sends the value of sum. Finally, it sends the newline character so that the next output will be in the new line. The multiple use of << in one statement is called *cascading.* When cascading an output operator, we should ensure necessary blank spaces between different items. Using the cascading technique, the last two statements can be combined as follows:

```
cout << "Sum = " << sum << "\n";
   << "Average = " << average << " \n";;
```

This is one statement but provides two lines of output. If you want only one line of output, the statement will be:

```
cout << "Sum = " << sum << ","
   << "Average = " << average << "\n";
```

The output will be:

```
Sum = l4, Average = 7
```

We can also cascade input operator >> as shown below:

```
cin >> numberl >> number2;
```

The values are assigned from left to right. That is, if we key in two values, say, 10 and 20, then 10 will be assigned to numberl and 20 to number2.

2.5 An Example with Class

One of the major features of C++ is classes. They provide a method of binding together data and functions which operate on them. Like structures in C, classes are user-defined data types.

Program 2.3 shows the use of class in a C++ program.

Program 2.3 Use of Class

```
#include <iostream>
using namespace std;
class person
{
    char name[30];

    int age;

    public:
        void getdata(void);
        void display(void);
};
void person :: getdata(void)
{
    cout << "Enter name: ";
    cin >> name;
    cout << "Enter age: ";
    cin >> age;
}
void person :: display(void)
{
    cout << "\nName: " << name;
    cout << "\nAge: " << age;
}

int main()
{
    person p;

    p.getdata();
    p.display();
```

```
        return 0;
    }
```

The output of Program 2.3 would be:

```
Enter Name: Ravinder
Enter Age: 30

Name: Ravinder
Age: 30
```

NOTE: ***cin*** *can read only one word and therefore we cannot use names with blank spaces.*

The program defines **person** as a new data of type class. The class person includes two basic data type items and two functions to operate on that data. These functions are called **member functions.** The main program uses **person** to declare variables of its type. As pointed out earlier, class variables are known as *objects.* Here, p is an object of type **person.** Class objects are used to invoke the functions defined in that class. More about classes and objects is discussed in Chapter 5.

2.6 Structure of C++ Program

As it can be seen from the Program 2.3, a typical C++ program would contain four sections as shown in Fig 2.3. These sections may be placed in separate code files and then compiled independently or jointly.

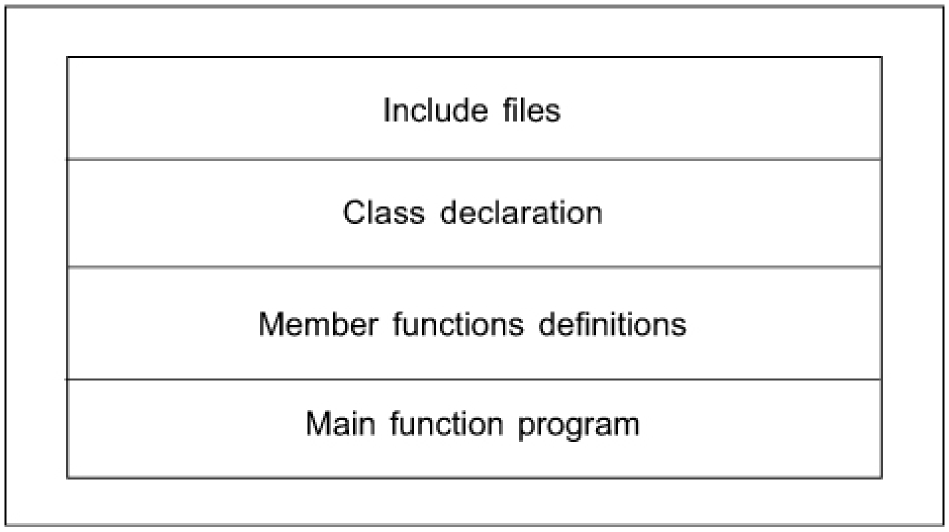

Fig. 2.3 *Structure of a C ++ program*

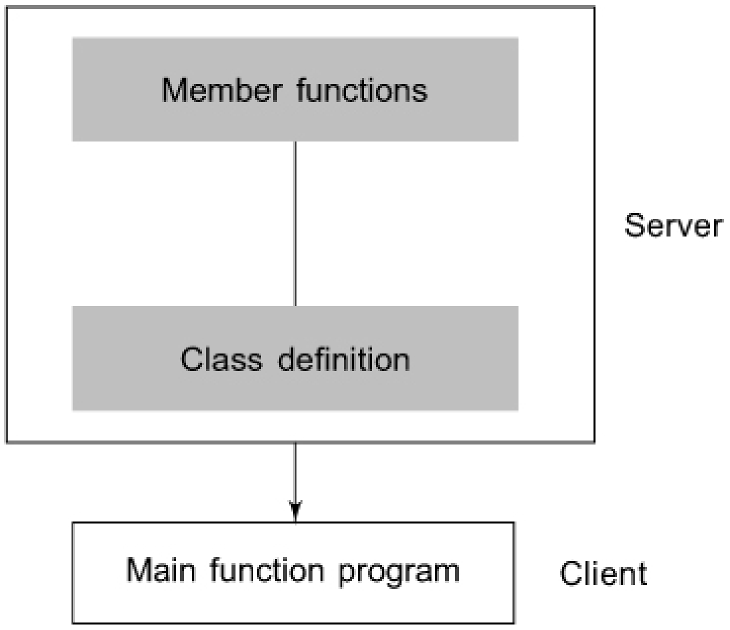

Fig. 2.4 *The client-server model*

It is a common practice to organize a program into three separate files. The class declarations are placed in a header file and the definitions of member functions go into another file. This approach enables the programmer to separate the abstract specification of the interface (class definition) from the implementation details (member functions definition). Finally, the main program that uses the class is placed in a third file which “includes” the previous two files as well as any other files required.

This approach is based on the concept of client-server model as shown in Fig 2.4. The class definition including the member functions constitute the server that provides services to the main program known as client. The client uses the server through the public interface of the class.

2.7 Creating the Source File

Like C programs, C++ programs can be created using any text editor. For example, on the UNIX, we can use vi or *ed* text editor for creating and editing the source code. On the DOS system, we can use *edlin* or any other editor available or a word processor system under non-document mode.

Some systems such as Turbo C++ provide an integrated environment for developing and editing programs. Appropriate manuals should be consulted for complete details.

The file name should have a proper file extension to indicate that it is a C++ program file. C++ implementations use extensions such as .c, .C, .cc, .cpp and .cxx. Turbo C++ and Borland C++ use .c for C programs and .cpp (C plus plus) for C++ programs. Zortech C++ system uses .cxx while UNIX AT&T version uses .C (capital C) and .cc. The operating system manuals should be consulted to determine the proper file name extensions to be used.

2.8 Compiling and Linking

The process of compiling and linking again depends upon the operating system. A few popular systems are discussed in this section.

Unix AT&T C++

The process of implementation of a C++ program under UNIX is similar to that of a C program. We should use the “CC” (uppercase) command to compile the program. Remember, we use lowercase “cc” for compiling C programs. The command

```
CC example.C
```

at the UNIX prompt would compile the C++ program source code contained in the file **example.C.** The compiler would produce an object file **example.o** and then automatically link with the library functions to produce an executable file. The default executable filename is **a.out.**

A program spread over multiple files can be compiled as follows:

```
CC fileLC file2.o
```

The statement compiles only the file **file1.C** and links it with the previously compiled **file2.o** file. This is useful when only one of the files, needs to be modified. The files that are not modified need not be compiled again.

Turbo C++ and Borland C++

Turbo C++ and Borland C++ provide an integrated program development environment under MS DOS. They provide a built-in editor and a menu bar which includes options such as File, Edit, Compile and Run.

We can create and save the source files under the **File option,** and edit them under the **Edit option.** We can then compile the program under the **Compile option** and execute it under the **Run option. The Run option** can be used without compiling the source code. In this case, the **RUN** command causes the system to compile, link and run the program in one step. Turbo C++ being the most popular compiler, creation and execution of programs under Turbo C++ system are discussed in detail in Appendix B.

Visual C++

It is a Microsoft application development system for C++ that runs under Windows. Visual C++ is a visual programming environment in which basic program components can be selected through menu choices, buttons, icons, and other predetermined methods. Development and execution of C++ programs under Windows are briefly explained in Appendix C.

Summary

- ❑ C++ is a superset of C language.
- ❑ C++ adds a number of object-oriented features such as objects, inheritance, function overloading and operator overloading to C. These features enable building of programs with clarity, extensibility and ease of maintenance.
- ❑ C++ can be used to build a variety of systems such as editors, compilers, databases, communication systems, and many more complex real-life application systems.
- ❑ C++ supports interactive input and output features and introduces a new comment symbol // that can be used for single line comments. It also supports C-style comments.
- ❑ Like C programs, execution of all C++ programs begins at **main( )** function and ends at **return( )** statement. The header file **iostream** should be included at the beginning of all programs that use input/output operations.
- ❑ All ANSI C++ programs must include **using namespace std** directive.

❑ A typical C++ program would contain four basic sections, namely, include files section, class declaration section, member function section and main function section.

❑ Like C programs, C++ programs can be created using any text editor.

❑ Most compiler systems provide an integrated environment for developing and executing programs. Popular systems are UNIX AT&T C++, Turbo C++ and Microsoft Visual C++.

Key Terms

#include **|** **a.out** **|** Borland C++ **|** cascading **|** **cin** **|** class **|** client **|** comments **|** **cout** **|** edlin **|** extraction operator **|** **float** **|** free-form **|** get from operator **|** input operator **|** insertion operator **|** **int** **|** **iostream** **|** **iostream.h** **|** keyboard **|** **main()** **|** member functions **|** MS-DOS **|** **namespace** **|** object **|** operating systems **|** operator overloading **|** output operator **|** put to operator **|** return () **|** screen **|** server **|** Simula67 **|** text editor **|** Turbo C++ **|** Unix AT&T C++ **|** **using** **|** Visual C++ **|** Windows **|** Zortech C++

Review Questions

2.1 State whether the following statements are TRUE or FALSE.

(a) Since C is a subset of C++, all C programs will run under C++ compilers.

(b) In C++, a function contained within a class is called a member function.

(c) Looking at one or two lines of code, we can easily recognize whether a program is written in C or C++.

(d) In C++, it is very easy to add new features to the existing structure of an object.

(e) The concept of using one operator for different purposes is known as operator overloading.

(f) The output function printf() cannot be used in C++ programs.

2.2 Why do we need the preprocessor directive #include <iostream> ?

2.3 How does a main() function in C++ differ from main() in C?

2.4 What do you think is the main advantage of the comment // in C++ as compared to the old C type comment?

2.5 Describe the major parts of a C++ program.

Debugging Exercises

2.1 Identify the error in the following program.

```
#include <iostream.h>
void main()
{
    int i = 0; i = i + 1;
    cout << i << " ";
     /*comment\*//i = i + 1;
    cout << i;
}
```

2.2 Identify the error in the following program.

```
#include <iostream.h>
void main()
{
     short i=2500, j=3000;
```

```
        cout >> “i + j = “ >> -(i+j);
    }
```

2.3 What will happen when you run the following program?

```
#include <iostream.h>
void main()
{
  int i=10, j=5;
  int modResult=0;
  int divResult=0;

  modResult =i%j
  cout << modResult << “ “;

  divResult = i/modResult;
  cout << divResult;
}
```

2.4 Find errors, if any, in the following C++ statements.

(a) cout << “x=“ x;

(b) m = 5; // n = 10; // s = m + n;

(c) cin >>x; >>y;

(d) cout << \n “Name:” << name;

(e) cout <<“Enter value:”; cin >> x;

(f) /*Addition*/ z = x + y;

Programming Exercises

2.1 Write a program to display the following output using a single cout statement.

Maths = 90
Physics = 77
Chemistry = 69

2.2 Write a program to read two numbers from the keyboard and display the larger value on the screen.

2.3 Write a program that inputs a character from keyboard and displays its corresponding ASCII value on the screen.

2.4 Write a program to read the values of a, b and c and display the value of x, where

$$x = a / b - c$$

Test your program for the following values:

(a) a = 250, b = 85, c = 25

(b) a = 300, b = 70, c = 70

2.5 Write a C++ program that will ask for a temperature in Fahrenheit and display it in Celsius.

2.6 Redo Exercise 2.5 using a class called **temp** and member functions.